10. Password Cracker

Write a Ruby script that accepting two args `hashed_password` `dictionary_file` to perform a dictionary attack to crack a hashed password.

Word 1	SHA-256 Hash 2
password	5e884898da28047151d0e56f8dc6292773603d0d6aabbddf554a4febe4e60ab3
———	———
123456	8d969eef6ecad3c29a3a629280e686cff8fabd2dfd771eebc9bc1696ec3e9d39
———	———
qwerty	d8578edf8458ce06fbc5bb76a58c5ca4a0a8cdd3d44f1e01c8a160d7038b1a5c
———	———
admin	8c6976e5b5410415bde908bd4dee15dfb16a4d96a35b7c4d465f9c74879ab0d4
———	———
welcome	25f9e794323b453885f5181f1b624d0b004d6e1d78d9e8a74b1e21610029a450
———	———

```
┌──(imenⓀhbtn-lab)-[…/scripting_cyber/0x00-ruby_scripting]
└─$ cat dictionary.txt
password
123456
qwerty
admin
welcome
```

```
┌──(imenⓀhbtn-lab)-[…/scripting_cyber/0x00-ruby_scripting]
└─$ ruby 10-password_cracked.rb
8c6976e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a81f6f2ab448a918
dictionary.txt
Password found: admin
┌──(imenⓀhbtn-lab)-[…/scripting_cyber/0x00-ruby_scripting]
└─$ ruby 10-password_cracked.rb
76e5b5410415bde908bd4dee15dfb16a4d96773603d0d6aabbddf dictionary.txt
Password not found in dictionary.
```

```
┌──(imenⓀhbtn-lab)-[…/scripting_cyber/0x00-ruby_scripting]
└─$ruby 10-password_cracked.rb
```

```
Usage: 10-password_cracked.rb HASHED_PASSWORD DICTIONARY_FILE
```

Script: `10-password_cracked.rb-`

```ruby
require 'digest'

def dictionary_attack(hashed_password, dictionary_file)
  # Read the dictionary file line by line
  File.foreach(dictionary_file) do |word|
    word.strip! # Remove any trailing whitespace or newline characters
    # Hash the word using SHA-256
    hashed_word = Digest::SHA256.hexdigest(word)
    # Compare the generated hash with the provided hash
    if hashed_word == hashed_password
      puts "Password found: #{word}"
      return
    end
  end
  # If no match is found
  puts "Password not found in dictionary."
end

# Main script execution
if ARGV.length != 2
  puts "Usage: #{File.basename(__FILE__)} HASHED_PASSWORD DICTIONARY_FILE"
else
  hashed_password = ARGV[0]
  dictionary_file = ARGV[1]

  # Check if the dictionary file exists
  unless File.exist?(dictionary_file)
    puts "Error: Dictionary file '#{dictionary_file}' not found."
    exit
  end

  dictionary_attack(hashed_password, dictionary_file)
end
```

How It Works:

1. **Imports the Digest Module**:
   - The `Digest` module provides the `SHA256` class for generating SHA-256 hashes.

2. **Reads the Dictionary File**:
    - The script iterates over each line in the dictionary file.
    - Each word is stripped of whitespace or newline characters.
3. **Hashes and Compares**:
    - The `Digest::SHA256.hexdigest` method generates the hash of the word.
    - If the hash matches the provided hashed password, the script prints the cracked password and exits.
4. **Handles Missing Files and Arguments**:
    - The script checks if the dictionary file exists.
    - Prints usage instructions if the required arguments are not provided.
5. **Command-Line Arguments**:
    - The script expects two arguments:
        1. `hashed_password`: The SHA-256 hash to crack.
        2. `dictionary_file`: The file containing potential passwords.

Example Usage:

1. **Missing Arguments**:

```
$ ruby 10-password_cracked.rb
Usage: 10-password_cracked.rb HASHED_PASSWORD DICTIONARY_FILE
```

2. **Password Found**:

```
$ ruby 10-password_cracked.rb
8c6976e5b5410415bde908bd4dee15dfb16a4d96a35b7c4d465f9c74879ab0d4
dictionary.txt
Password found: admin
```

3. **Password Not Found**:

```
$ ruby 10-password_cracked.rb
76e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a81f6f2ab448a918
dictionary.txt
Password not found in dictionary.
```

4. **Dictionary File Not Found**:

```
$ ruby 10-password_cracked.rb
8c6976e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a81f6f2ab448a918
missing_dictionary.txt
Error: Dictionary file 'missing_dictionary.txt' not found.
```

Explanation of Tools:

- `Digest::SHA256.hexdigest`:
  Generates the SHA-256 hash of a given string.
- `File.foreach`:
  Reads the dictionary file line by line for memory efficiency.
- `ARGV`:
  Used to handle command-line arguments.

This script is efficient, handles errors gracefully, and follows good practices for cracking hashed passwords using a dictionary attack.